Migrations

- How to move databases in an Informix instance from one server to another
  - New server
  - New O/S
  - New disks
  - New database technology

Migrations

- High level approach
- Steps
- Summary of tools
- Considerations & Complications
- dbexport/dbimport

Similar Environments

- If moving to similar hardware & similar OS & same Informix version…

Similar Environments – backup/restore

- Install informix on target, or copy installation
- Create files or LVs for chunks
  - Do not need to create chunks/dbspaces
- Archive on source, restore to target (redirected restore)

Similar Environments - Copy Chunks

- Install informix on target, or copy installation
- Block or shutdown source instance
- Copy chunks & links to target:
  - Copy cooked files to target, or
  - Perform SAN copy

Similar Environments - ifxclone

- Install informix on target, or copy installation
- Use ifxclone on source to copy instance to target
  - Can use **`--createchunkfile`** option to prevent the need for creating dbspace files

New SAN

Informix 12.10+ Only!

- Create devices on new array
- Mirror the chunks to the new array
- Allow to sync
- Switch primary and mirror chunks

```
execute function task("modify chunk
swap_mirror",<chunk number>)
```

or

```
execute function task("modify space
swap_mirrors","<space name>")
```

Different Environments

- Differences between hardware, OS, or Informix versions prevents copies of the chunks or backup/restore approach
- Need to perform a copy of the data itself in ASCII format
  - Unload & Load

Copying to a New Environment

- Install & configure Informix on the target
- Create LVs/files, dbspaces, etc on the target
- Create databases on target
- Create tables on target
- Unload data from source
- Copy data to target
- Load data on target
- Create views, indexes, constraints, users, stored procedures, permissions, synonyms
- Update Statistics

Prepare the Target Environment

- Install Informix on target - can be different version
- Create LVs or cooked files - same or different
- Create links to chunks
- Copy onconfig and review, or update standard version if upgrading
- Copy sqlhosts file, review and update
- Update hosts, services, trust files are correct
- Update ALARMPROGRAM, evidence.sh
- Create users/groups on target

Prepare the Target Environment

- Initialize Informix
- Create logical logs
- Move physical log
- Create chunks/dbspaces
- Create databases
- Create row types
- Create TimeSeries containers

```
dbschema -c -ns
```

dbschema -c

- Outputs the SQL statements to:
  - Create chunks & dbspaces
  - Create logical logs
- Use the -ns option to output commands (onspaces, onparams, etc)
- Run dbschema -c on the ***source*** instance
- Execute output on the ***target*** instance

dbschema -c -ns

```
# Dbspace 1 -- Chunk 1
# onspaces -c -d rootdbs -k 2 -p /informix_chunks/rootdbs.1 -o 0 -s 80000 -ef
500 -en 400

# Dbspace 2 -- Chunk 2
onspaces -c -d physdbs -k 2 -p /informix_chunks/physdbs.1 -o 0 -s 50000 -ef 100
-en 100
.
.
# Physical Log
onparams -p -s 49800 -d physdbs -y
.
.
# Logical Log 4
onparams -a -d logdbs -s 10000

# Logical Log 5
onparams -a -d logdbs -s 10000
.
.
dbaccess sysadmin << END
SELECT TASK ('drop log', log) FROM sysadmin:llog
WHERE sysmaster:bitval(flags,'0x02')==0;
END
.
.
```

Create Databases

- Which DBSpace should hold the database?
- If not specified, databases will be created in the root dbspace
- Identify the dbspace and logging mode used on the source system
- If source database is in the rootdbs, then the migration allows the database to be relocated on the target

Database DBSpace

```
database sysmaster;

select name, owner,
dbinfo("DBSPACE",partnum) dbspace,
is_logging,is_buff_log
from sysdatabases
where name not matches "sys*";
```

```
name          stores_demo
owner         informix
dbspace       datadbs
is_logging    1
is_buff_log   0
```

Create Database

- Log in as the owner of the new database
- Make sure that you have the correct database owner!! It cannot be changed later!

```
database sysmaster;
create database <dbname> in <dbspace>;
```

Other options for logging, ANSI, case sensitive

DBCREATE_PERMISSION may limit who can create databases

Prepare Target Schemas

- Get the source database schemas using **dbschema**
- For ALL objects in the database:

```
dbschema –d <dbname> -ss <output_file>.sql
```

- **-ss** option will preserve dbspaces names, extent sizes and lock modes (row or page)
- ***output_file.sql*** will contain SQL definitions of tables, indexes, stored procedures, permissions, synonyms, etc.

Prepare Target Schemas

- Split out the file into multiple files by object type
- Create separate files for:

Object Type	dbschema options (-d <dbname> implied)
Tables/Indexes/Views/Triggers	-t all -ss
Synonyms	-s all
Sequences	
User Defined Types	-u all
Stored Procedures	-f all
Roles	-r all
Permissions	-p all

Split out views & triggers into separate files

Prepare Target Schemas

Update SQL:

- Different dbspaces
  - Split out indexes
  - Page size
- Extent sizes
- Lock mode
- Other…?

Create the Tables & Indexes

Run the SQL to create the tables & indexes:

```
dbaccess -e <target db> <create_table>.sql >
<create_table>.out 2>&1
```

e.g:

```
dbaccess -e mydb tables.sql > tables.out 2>&1
```

Review output for errors

```
egrep "^ *[0-9][0-9]*:" tables.out

 9628: Type (address_t) not found.
  206: The specified table (informix.employee) is not
in the database.
  111: ISAM error:  no record found.
```

Unloading/Loading the Data

- For non-Informix databases as the source, use appropriate utilities to unload data in ASCII format, and preferably pipe delimited

- For small databases, dbexport/dbimport is simple
  - Do not need to create the database or tables beforehand

Unloading/Loading the Data

- For larger databases, will likely use a ***combination*** of tools & utilities:
  - Modified dbexport/dbimport
  - dbschema
  - dbaccess
  - SQL unload/load
  - External tables
  - dbload
  - High Performance Loader (HPL)
  - TimeSeries Loader (TSL)
  - Enterprise Replication (ER)
  - UNIX: gzip/compress, awk, sed, shell, sftp, pipes…

Final Steps

Create:

- Indexes
- Constraints
- Sequences
- Synonyms
- Views
- Stored procedures (No PDQ)
- Triggers
- Roles
- Permissions

Dependencies?

Update Statistics

Things to consider…

Indexes & Constraints

Do not load tables with Indexes or Constraints active - performance will suffer!

```
set constraints for <table> disabled;
set indexes for <table> disabled;

  …load table…

set indexes for <table> enabled;
set constraints for <table> enabled;
```

If needed:

```
set triggers for <table> [disabled|enabled]
```

Logging

- When possible, the target database should use “no logging” when loading tables
  - Avoids long transactions
  - Reduces the number of locks used
  - Faster

Logging

- Cannot use no-logging if replicating target server (HDR/RSS/ER)
- Cannot create synonyms that reference external databases if one database uses logging and the other does not

```
568: Cannot reference an external database without logging.
569: Cannot reference an external database with logging.
```

  – Enable logging before creating these synonyms
- Do not use no-logging if loading TimeSeries

Logging

- Create database with no logging
- Disable/Enable logging with "**ondblog**"
- If needed, enable logging on target *after* tables have been loaded
  - Buffered
  - Unbuffered (recommended)

```
ondblog unbuf testdb
ondblog complete, returning 0 (0x00)
onbar -b -F
```

```
ondblog nolog testdb
ondblog complete, returning 0 (0x00)
```

Logging

- Disable logging on individual tables by setting them to RAW

```
alter table cust type (raw);
```

or set to RAW when the table is created

```
create raw table cust (id serial, ...)
```

- Enable logging after table is loaded

```
alter table cust type (standard);
```

Raw tables cannot have unique, PK or FK constraints
Not enough to simply disable them

Making Changes

A migration is a good opportunity to make some changes…

- Different page sizes
  - New BUFFERPOOL definitions
  - DBSpace definitions
- New or resized dbspaces
- Extent sizes
- Locking modes
- Partitioning (rolling windows?)
- Logging (no logging, buffered, unbuffered)
- TimeSeries (containers, origin, rolling windows)
- Configuration changes

Perform a thorough review of source system for performance bottlenecks, sizings, active tables, etc

Weigh: Risk vs Reward

Moving Data

How to get the data to the target?

- ftp/sftp/rcp
- NFS mount (configuration!)
- Compress/uncompress
- Space for staging files
- Max file size

Weigh the time to compress/uncompress against the time saved when moving smaller amounts over the network

Outage Window

How long have you got?

- Tuning the migration process is a major effort
- Test, Modify, Repeat
- Script & document
- Static vs volatile tables, and data within tables
- How to maintain referential integrity?
- Chop and parallelize - how much can you do at one time?
- Production freeze

Validation

How to check?

- Review log files
  - grep “number of rows loaded”
  - grep for errors
  - Check return value of each command
- Row counts
  - Use sysmaster:systabinfo.ti_nrows
- Sum column values
- TimeSeries - TSContainerNElems function
- User testing

dbexport/dbimport

- **Pros**
  - Utilities provide a simple way to move an entire database
  - Only requires a few steps
  - Easy to use
- **Cons**
  - Slow
  - Export locks source database

dbexport

- Dumps all schemas (tables, indexes, procedures, etc) to a single SQL file
- Loops through each table and unloads contents of each to a separate file
- All unloads in ASCII - pipe-delimited
- SQL and unloads in directory <*dbname*>.exp
- Can also export to tape

dbexport

```
dbexport <database> [-X] [-c] [-q] [-d] [-ss [-si]]
  [{ -o <dir> | -t <tapedev> -b <blksz> -s <tapesz>
[-f <sql-command-file>] }] [-nw]
  [-no-data-tables[=table name{,table name}]]
  [-no-data-tables-accessmethods[=access method
name{,access method name}]]
```

dbexport

```
dbexport stores_demo -ss
```

-ss option preserves dbspace names, extent sizes, lock level

- Creates output file **dbexport.out**
- Creates directory **stores_demo.exp**
- Contains SQL file and unload files

dbexport

- dbexport creates directory stores_demo.exp:

```
drwxr-xr-x 2 informix informix  4096 Apr 30 14:05 stores_demo.exp
-rw-r--r-- 1 informix informix 15829 Apr 30 14:05 dbexport.out
```

- stores_demo.exp contains SQL file and unload files:

```
-rw-r--r-- 1 informix informix  2515 Apr 30 16:49 custo00100.unl
-rw-r--r-- 1 informix informix  1735 Apr 30 16:49 order00102.unl
-rw-r--r-- 1 informix informix   151 Apr 30 16:49 manuf00103.unl
-rw-r--r-- 1 informix informix  2864 Apr 30 16:49 stock00104.unl
-rw-r--r-- 1 informix informix  1508 Apr 30 16:49 items00105.unl
-rw-r--r-- 1 informix informix   697 Apr 30 16:49 state00106.unl
-rw-r--r-- 1 informix informix    90 Apr 30 16:49 call_00107.unl
-rw-r--r-- 1 informix informix  1673 Apr 30 16:49 cust_00108.unl
-rw-r--r-- 1 informix informix 13639 Apr 30 16:49 catal00109.unl
-rw-r--r-- 1 informix informix   108 Apr 30 16:49 tab__00110.unl
-rw-r--r-- 1 informix informix   543 Apr 30 16:49 wareh00111.unl
-rw-r--r-- 1 informix informix    66 Apr 30 16:49 class00112.unl
-rw-r--r-- 1 informix informix   179 Apr 30 16:49 emplo00113.unl
-rw-r--r-- 1 informix informix    45 Apr 30 16:49 testt00115.unl
-rw-r--r-- 1 informix informix 15809 Apr 30 16:49 stores_demo.sql
```

dbexport – Skip Tables

- May not want to unload large tables with dbexport
- Skip named tables with

  [-no-data-tables[=table name{,table name}]]

- No unload file created
- Still creates the table in the SQL file

  The “unload” entry is NOT included in the SQL file

```
{ unload file name = item_00103.unl number of rows = 111543 }
```

dbexport - Failures

- Locking – a user is connected to the database

```
-425 - Database is currently opened by another user.
-107 - ISAM error:  record is locked.
```

- Insufficient space
- Export cannot be resumed after failure
- Directory already exists

```
Subdirectory already exists
```

dbimport

- Creates the database
- Loads tables serially
- Creates views, indexes, stored procedures, permissions, statistics, etc.
- Subject to long transactions and high number of locks (if using logging)

dbimport

```
dbimport <database> [-X] [-c] [-q] [-d <dbspace>]
  [-l [{ buffered }] [-ansi]] [-ci] [-nv] [-D]
  [{ -i <dir> | -t <tapedev> [ -b <blksz> -s <tapesz> ]
[-f <script-file>] }]
```

dbimport

```
dbimport stores_demo -d datadbs
```

-d option specifies the dbspace where the database is created. If not specified, database will be created in the root dbspace (not recommended)

- Will look for directory **stores_demo.exp**
- Will use the SQL file **stores_demo.sql**
- Unless otherwise specified, will create the database with no logging

dbimport – Modify SQL File

- The SQL file can be modified
  - Change DBSpace names
  - Change Extent Sizes
  - Change Lock Mode

dbimport – Change DB Name

- Import with different database name
- Rename the export directory *and* the SQL file

```
mv stores_demo.exp stores_demo2.exp
cd stores_demo2.exp
mv stores_demo.sql stores_demo2.sql
cd ..

dbimport stores_demo2 -d datadbs
```

dbimport – Failures

- Cannot be resumed after failure
  - Drop the database
  - Restart

- But, if the “loads” have completed…
  - Run the remaining SQL in dbaccess

- Data Errors:

```
Load file has different number of columns
than table
*** Import data is corrupted!
```

dbimport – Skip Tables

- Tables can be excluded from the import by modifying the SQL file
- Remove the “unload” comment from the file

```
{ TABLE "informix".cust_calls row size = 531 number of columns = 7 index size =
31 }

{ unload file name = cust_00108.unl number of rows = 7 }

create table "informix".cust_calls
  (
    customer_num integer,
    call_dtime datetime year to minute,
    user_id char(32)
        default user,
    call_code char(1),
    call_descr char(240),
    res_dtime datetime year to minute,
    res_descr char(240),
    primary key (customer_num,call_dtime)
  ) extent size 16 next size 16 lock mode row;
```

Remove

dbimport – Logging

- Recommend importing database with no logging and change to logging when complete

  `ondblog unbuf <dbname>; onbar -b -F`

- Create logged database with dbimport with:

  `[-l [{ buffered }]`

- Remove synonyms that reference external databases and run *after* logging enabled
- Modify the SQL file to create table as RAW, and alter to STANDARD *before* primary and foreign keys, and unique constraints
  - May require changing primary key definitions